REMARKS

Claims 1 – 20 are pending and stand rejected as of the First Office Action.

Please amend the Title, Description, and Claims as shown above.

The title has been amended to append the phrase "for allocating computer resources" in response to the objection in paragraph 3 of the Office Action.

In paragraphs 11, 13, 19, and 110, the term "master job dispatch queue" has been consistently applied to distinguish "master job dispatch queue," which is produced by sorting, from "master job queue", which a simple list or a filtered list, but not a sorted list.

In paragraphs 12, 13, 28, 62, 63, 66, 67, 68, 70, 71, 74, 75, 76, 103, 104, 106, 108, 109, 112, 113, 115, 126, and 144, grammatical and typographical errors have been corrected.

In paragraph 19, 20, 62, 66, 68, 71, 74, 75, 82, 89, 90, 91, 95, 104, 113, 119, 123, 125, 127, 131, and 146, parenthetical, appositive, or declarative statements have been added that are either well known in the art (i.e., paragraph 19) or have been copied from later parts of the Description and moved forward to improve clarity (i.e., paragraphs 20, 62, 66, 68, 71, 74, 75, 82, 89, 90, 91, 95, 104, 113, 119, 123, 125, 127, 131, and 146). The majority of the clarifications are reminders of the terminology used to distinguish OOP vs. non-OOP embodiments of the invention, e.g., job type objects vs. job type elements, and job containers vs. job type directories; most of the remaining clarifications are reminders of the system-defined limits, n/2 and m/2, of filtered matching before running a sort.

In paragraph 71, a copy and paste error (source text at the end of paragraph 67) has been corrected.

Paragraph 80 has been amended to delete a hyperlink in response to the objection in paragraph 2 of the Office Action.

Paragraph 121 has been amended to include an inadvertently omitted second line in the specific example of a trigger definition, so that the Example conforms to the explanation and generic example in paragraph 120.

Claims 1 and 6 have been amended to start each step with a lower case letter in response to the objection in paragraph 4a of the Office Action.

The term "EDQS" is defined in the first sentence of paragraph 19 of the as-filed application: "The Event-Driven Queuing System and Method ("EDQS") invention uses a continually updated, event-driven, supervisory daemon together with one or more callback, event, process groups, job typing, message architecture, clients, workers, executors, and related functions, as described below, to solve the above-described problems in the existing art." Paragraph 19 is the first usage of the term EDQS in the application.

The terms "GUI" is defined in the second sentence of paragraph 23 of the as-filed application: "Input from end-users who submit jobs for processing, and input from system administrators, is collected in an EDQS graphical user interface ("GUI") and translated by the EDQS messaging application program interface ("MAPI") into a message containing an EDQS DSL statement." Paragraph 23 is the first usage of the term GUI in the application.

Applicant requests that the Examiner withdraw the objection in paragraph 4/4b of the Office Action concerning the terms EDQS and GUI.

In the second paragraph "2" in the Office Action (actually paragraph "5"), the Office objects to claims 13 and 17 – 20:

> "as being of improper dependent form for failing to further limit the subject matter of a queuing system defined in claim 12. Applicant is required to cancel the claim(s), or amend the claim(s) to place the claim(s) in proper dependent form, or rewrite the claim(s) in independent form. The job type data architecture and event/callback architecture as defined in claims 13 and 17 - 20 fail to further limit the queuing system of claim 12."

Applicant notes that only claim 13 depends from claim 12. Claims 17 – 20 depend from claim 16, which in turn depends from claim 11. As to claim 13, Applicant respectfully suggests that claim 13 is in proper dependent form: claim 12 recites a job type element comprising several listed elements, but the elements listed in claim 12 do not include "code files identified in the job type element". Claim 13 further limits claim 12 by specifying that the "job type directory that contains the job type element" also contains "code files identified in the job type element".

As to dependent claims 17 – 20, each of which depend from claim 16, which in turn depends from claim 11, claims 11 and 16 do not recite a limitation of "job-driven dispatch" (which is added as a limitation in claim 17), "worker-driven dispatch" (which is added as a limitation in claim 18), "trigger evaluat[ion of] Boolean expressions of event states of one or more jobs in a process group" (which is added as a limitation in claim 19), and the use of

"evname, evtype, and evcontext variables to define an event" (which is added as a limitation in claim 20). Applicant respectfully suggests that claims 17 – 20 are in proper dependent form.

Applicant respectfully requests that the Examiner withdraw the objection in paragraph "5" of the Office Action as to the form of dependent claims 13 and 17 – 20.

Claim Rejections: 35 USC § 101

In paragraph "4" of the Office Action, the Office rejected claim 11, and claims 12 to 20 that depend from claim 11, as "being directed to non-statutory subject matter" under 35 U.S.C. § 101.

To provide guidance on software claims in the light of 35 U.S.C. § 101, MPEP 2106.1 states:

> Computer programs are often recited as part of a claim. USPTO personnel should determine whether the computer program is being claimed as part of an otherwise statutory manufacture or machine. In such a case, the claim remains statutory irrespective of the fact that a computer program is included in the claim. The same result occurs when a computer program is used in a computerized process where the computer executes the instructions set forth in the computer program. Only when the claimed invention taken as a whole is directed to a mere program listing, i.e., to only its description or expression, is it descriptive material *per se* and hence nonstatutory. * * * When a computer program is claimed in a process where the computer is executing the computer program's instructions, USPTO personnel should treat the claim as a process claim.

The first sentence of amended paragraph [3] of the Application now reads:

> "A "queuing system" is software <u>running on computer resources</u> that allocates the processing of jobs among networked computers, issues commands to such networked computers, and tracks the jobs through completion; many queuing systems generate status and/or error reports as part of job tracking."

The Office acknowledged the definition of "queuing system" in the first sentence of paragraph [3] of the Application as the basis for construing the preamble of claim 11. The first sentence of paragraph [3] is read in the context of the remainder of paragraph [3] and later paragraphs. Paragraph [3] of the Application continues, "Queuing systems exchange messages with users and with processes running on resources, such as computers, storage devices,

networks, network devices, and associated software (collectively, "resources").", and paragraph [4] discloses "The "management daemon" (called a "supervisory daemon" in the invention) in a queuing system is the executive component, as distinct from services that are provided by resources in response to commands of the executive. A "worker daemon" is a process running on a resource that causes jobs to be processed on a worker in response to messages from a management daemon." Paragraph [20] states, "The term "supervisor" means a host that runs, among other software, a "supervisory daemon," as explained below. The term "worker" means a host on which a job is processed. A worker in the EDQS invention runs, among other software, a "worker daemon" and one or more "executors," as explained below." The quoted parts of paragraphs [3], [4], and [20], *inter alia*, support the amendment of the first sentence of paragraph [3] that a queuing system runs on computer resources: each supervisory daemon runs on a supervisor computer, each worker daemon runs on a worker, and each executor runs on a worker.

Applicant has amended the first sentence of paragraph 1 to insert in the definition of queuing system the phrase "running on computer resources" to make express the inherent, and disclosed, structure that queuing systems comprise software running on computer resources, which computer resources execute the programming code of the queuing software. This definition of queuing system then applies, as the Office notes, to the preamble of claim 11. A system that "allocates the processing of jobs among networked computers, issues commands to such networked computers, and tracks the jobs through completion" is thus tangibly embodied in a manner so as to be executable and produces a "useful, tangible, and concrete result", as required by the holding of *State Street,* 149 F.3d at 1373-74, 47 USPQ2d at 1601-02.

In view of the amendment of paragraph [3] of the Description, which informs the construction of the preamble of claim 11, Applicant respectfully requests that the rejection of claims 11 – 20 be withdrawn.

Claim Rejections - 35 USC § 102

In paragraph "5" of the Office Action, the Office rejected claim 1 - 10 under 35 U.S.C. 102(e) as being anticipated by U.S. Patent Application Publication No. 2003/0154112 to Neiman et al. ("Neiman").

Anthony Higa is one of the inventors of the EDQS invention. The Declaration of Anthony Higa, submitted herewith, establishes the EDQS invention "was initially conceptualized

no later than 1 December 1999, was fully conceptualized no later than 21 August 2001, and was thereafter diligently reduced to practice through the filing date of 23 January 2004." The filing date of Nieman is 29 October 2002. Therefore, the date of invention of the EDQS invention was no later than 21 August 2001, which predates the filing date of Nieman by more than 14 months. Therefore, Nieman is disqualified as a prior art reference under 35 U.S.C. 102(e).

Notwithstanding the disqualification of Nieman under 35 U.S.C. 102(e), and without waiver of such disqualification of Nieman, the following comparison of Nieman versus the EDQS invention establishes that Nieman does not anticipate the EDQS invention.

Neiman does not disclose how his scheduling system operates to match jobs and nodes, and therefore cannot anticipate the EDQS queuing system. Nieman is directed to reserving time slots during which jobs will run, and allows a job request to state a preference for time of job execution, which "node" (equivalent to a "worker" in the EDQS invention) will perform a job, a suggested priority, minimum hardware requirements, etc. (Neiman, paragraphs [0044 - 0045]). Each embodiment of Neiman includes "a central queue 500, [and] a scheduler 600" (Neiman, paragraph [0064]). The complete, detailed description of Neiman's queue 500 and scheduler 600 is eight paragraphs, [0071 -0073] and [0075 – 0079] respectively, but Neiman omits an explanation in these paragraphs, as well as in the entire patent application, of how jobs are matched with "nodes" (i.e., workers). Neiman's scheduler operates on a "first-come, first-served" basis: "Each job 182-1 to 182-N is stored in the queue 500 prior to processing (step 1625)." (Neiman, paragraph [0122]) Nieman alludes to a "scheduling algorithm" (Nieman paragraph 0075), but does not disclose what it is or how it matches jobs and workers.

Although Neiman mentions "the scheduler 600 may use policy and priority rules to allocate, for a particular session, the resources of multiple CPUs in a pool of node computers 800" (Neiman, paragraph [0075]), Neiman omits how policy and priority rules are actually implemented. *Matching of jobs and workers is the essence of a scheduling system.* Neiman's queue is a database of the chronological order of job submission in which job results can be associated with job requests (Neiman, paragraph [0071]). Neiman's "queue 500" is solely concerned with jobs, not with workers. The existing art of scheduling systems uses a job-driven, periodic sort, so in the absence of any disclosure to the contrary, Neiman's queue, scheduler, and scheduling algorithm are either enabled by a well-known method (job-driven, periodic sort), or are not enabled at all. In contrast, the EDQS Application describes not only how existing art

queuing systems work, but how the event-driven nature of the EDQS invention differs substantively from existing art systems.

Neiman does not disclose how heterogenous platforms communicate over a network without building unique messages for the API on each type of user and worker platform. Simply stating that an API on each user, supervisor, and worker exchanges messages using a transparent network protocol, "e.g., SOAP, XML/HTTP or its variants" (Neiman, paragraph [0051]) does not explain how a given type of job, with specific requirements for job execution, communicates those requirements to an unknown worker. The exact worker that will execute a job is presumed to be known to the API of the calling application.

Neiman does not disclose how his scheduling system handles long-duration jobs or scales up for short-duration jobs. Paragraph [0072] of Neiman states, "For normal load conditions in the compute backbone 300 infrastructure of one embodiment, the time it takes to receive a request, send it to a node computer 800, and retrieve the result should take no more than 500 ms, with 100 ms or less being optimal." (Neiman, paragraph [0072]) If job execution time overruns arose, or a node initially denied a request, Neiman simple sends the denied job to the back of the queue. Paragraph [0078] of Neiman states, "The scheduler 600 also may communicate with the service manager 700 to take appropriate action when a node computer 800 becomes unavailable due to failure, reassignment for use by another service, suspension, or other reason. In such cases, the scheduler 600 reschedules computations running on the failed or reassigned node computer 800 so that the results from all jobs 182-1 to 182-N sent to the compute backbone 300 are eventually completed and returned to the appropriate calling application 180." However, Neiman does not define or disclose "appropriate action", e.g., how a denied job is matched with a worker. Neiman, in paragraph [0119], does disclose, "In the case of a failure (i.e., the computation was not completed) an error indication may be returned in place of the task output 189." Neiman is expressly designed for short duration jobs. Sending a short duration or time-critical jobs to the back of the queue is problematic or even fatal if dispatch delays are long (dispatch delays grow exponentially with increases in the population of jobs and workers); Neiman cannot scale up because it produces exponential increase in dispatch time. Providing an "error indication" instead of task output is inadequate as an "appropriate action". In contrast, the EDQS Application handles both short-duration and long-duration jobs, and provides only arithmetic increases in delay as the population of jobs and workers increase.

Barroux discloses a true/false test, not a scheduling system that match jobs and workers, and does not handle job failures; Barroux, therefore, cannot render the EDQS queuing system obvious. Barroux "is a tool for collecting and managing survey information about nodes of network 202". (Barroux, discussion of Figure 2). Barroux discloses an interesting use of remote procedure calls, but his "task scheduler" is literally "first come, first served" without sorting or filtering. Barroux describes his "task scheduler" as follows: "Upon receiving the message, task scheduler 302 checks the time against the exclusion periods for the target node at step 512. If the message's time is excluded, task scheduler 302 discards the message at step 514. If the message's time is not excluded for the target node, task scheduler 302 proceeds to step 516 where exclusion periods for the subnet to which the target node belongs are checked. If the message's time is excluded for the subnet, task scheduler 302 discards the message at step 514. If the message's time is not excluded, task scheduler 302 proceeds to step 518 where exclusion periods specified for the whole network are checked. If the message's time is excluded for the whole network, again task scheduler 302 discards the message at step 514. If the message's time is not excluded at any level, at step 520 it passes to ProcLoad module 306 for launching of the task identified in the message."

In other words, Barroux' "task scheduler" is a three-step Boolean test for whether a previously scheduled job exists for a given time slot: time slot availability is tested at the node, subnet, and network levels. Barroux does not mention the possibility of job failure after a job (in Barroux, a "task") is scheduled, much less how job failure is remedied. In contrast, the EDQS Application describes not only how far more sophisticated existing art queuing systems work, but how the event-driven nature of the EDQS invention differs substantively from existing art systems. Barroux' elementary test for time slot availability would not suggest or imply an event-driven queuing system as disclosed in the Application.

Barroux does not disclose how his scheduling system handles long-duration jobs or scales up for short-duration jobs. Barroux is expressly designed for short duration jobs, i.e., asset survey queries and responses, and discards requests that conflict with earlier requests. Barroux is limited by a prior reservation, even if the reserved time is not used. Therefore, Barroux cannot scale up because it discards conflicted requests and cannot exceed 24 "reservation hours" per day; it cannot dispatch a later job when a previous job completes early. In contrast, the EDQS Application handles both short-duration and long-duration jobs, provides

only arithmetic increases in delay as the population of jobs and workers increase, and can dispatch a later job when a previous job completes early.

The following table summarizes the inventive steps of the EDQS invention over existing art systems like those of Neiman and Barroux.

Function	Neiman	Barroux	EDQS	Comment
Event-driven	No	No	Yes	Both Neiman and Barroux use job-driven queues, without consideration of worker attributes. Neither Neiman or Barroux address event-driven functionality, triggers, or callbacks.
Express disclosure of matching job and worker	No	No	Yes	Neiman doesn't disclose how jobs are matched with "nodes" (workers), and Barroux simply tests for earlier reservations.
Filtered job queue / filtered worker queue, and optional sort	No	No	Yes	Neither Neiman nor Barroux filter jobs or workers. Neiman does not disclose how jobs are matched with nodes (workers). Filtering before sort in queuing systems is novel to EDQS.
Communication among heterogenous platforms without API-specific messages	No	No	Yes	Neiman mentions message exchange among different APIs, but does not explain how message protocols such as SOAP and XML/HTTP are used to do so. Barroux does not address the function. The EDQS invention uses DSL.
Check environment process	No	No	Yes	Neiman discloses routing related to amount of resources. Barroux does not address the function. Check environment process, in which a worker confirms that it can run a job, in queuing systems is novel to EDQS.
Worker-driven	No	No	Yes	Neiman and Barroux do not disclose worker-driven job dispatch. Worker-driven worker/job matching in queuing systems is novel to EDQS.
EDQS messaging architecture				Neiman and Barroux disclose network messaging known in the art (SOAP, XML/HTP), but, unlike EDQS, do not disclose a messaging architecture that enables event/callbacks.

Callbacks				Neiman does not disclose conditional branching or other functions provided by the event/callbacks. Barroux does not address callbacks or the equivalent. Callbacks in queuing systems are novel to EDQS.
Job type data architecture	No	No	Yes	Neiman discloses including optional meta-information in a job request, but not how the meta-information is used in a data architecture, or meta-information sufficient to support events and callbacks. Barroux does not address the function. Job type data architecture in queuing systems is novel to EDQS.
Domain specific language	No	No	Yes	Neiman discloses hardcoding APIs to given types of calling applications, but not a domain specific language. Barroux does not address the function. Domain specific language in queuing systems is novel to EDQS.

To anticipate the claims in a later filed application, the prior art reference must use substantially the same structural elements in substantially the same way to produce substantially the same results. Neiman cannot anticipate the EDQS invention because it does not have substantially the same structural elements as the EDQS invention, cannot operate in substantially the same way, and cannot produce substantially the same results. Importantly, Nieman alludes to a “scheduling algorithm” (Nieman paragraph 0075), but does not disclose what it is or how it matches jobs and workers.

In paragraph “7” of the Office Action, the Office asserts that Neiman anticipates claim 1 of the Application. Neiman’s method of job dispatch, “allocate an appropriate amount of computing resources to particular jobs” (Neiman p. 7, paragraph 0075) does not *match jobs to workers*, it simply *allocates* without explaining how a worker is selected. Therefore, the exact worker that will execute a job is presumed to be known to the API of the calling application. “Deploying the worker modules 195-1 to 195-N on the compute backbone 300” [Neiman, p. 3, paragraph 0042] is not aliasing each worker in a plurality of workers to a same or different node, it is simply mounting workers on a network. Neiman’s “worker module” is analogous to a “job process” managed by an “executor” in the EDQS invention (Application, paragraph 56) and

Neiman's "node" is a "worker" in the EDQS invention. Neiman does not alias worker modules or anything else. Neiman lacks the structural elements of nodes and workers aliased to nodes, as defined in the EDQS invention. The "queue 500" [Neiman p. 7, paragraph 0078] is not a queuing system, it is a job list. "Meta-information supplied by a job 182 *may identify* more than one worker 155-1 to 155-N" (Neiman p. 3, paragraph 0044; the identity of each worker that will execute a job is therefore known to the API of the calling application) is not filtering the workers to produce an unsorted, eligible worker list: job meta-information is not worker attributes, it is information about jobs. If the job request from the API earmarks a specific worker, or set of workers, there is no need for a sophisticated job/worker matching process: *the worker specified in a job request is either available or not.* Neiman does not address worker attributes and capabilities, or whether jobs or workers are sorted, filtered, or otherwise matched. Neiman does not build an eligible worker list and therefore lacks a key structural element of the EDQS invention. "Schedules the jobs 182-1 to 182-N on available node computers 800-1 to 800-N" (Neiman pp. 11 - 12, paragraph 0122) is not searching the eligible worker list to match a job with a worker; Neiman does not disclose a worker list or how jobs are matched to workers, and therefore Neiman lacks a key structural element of the EDQS invention. "Determines availability of the node computers 800-1 to 800-N (step 1630), and schedules the jobs 182-1 to 182-N on available node computers 800-1 to 800-N" (Neiman pp. 11-12, paragraph 0122) is not a building an eligible worker list (available does not mean capable) and does not describe how jobs are matched to workers.

In paragraph "8" of the Office Action, the Office asserts that Neiman discloses removal of a worker, upon job failure, from the eligible worker list that anticipates claim 2 of the Application. Removing the worker from the "service allocated" is not removing the worker from the eligible worker list because Neiman does not construct an eligible worker list. The nodes (in the EDQS, "workers") of Neiman are either "available" or not, and therefore Neiman lacks a key structural element of the EDQS invention. Neiman does not disclose how jobs and workers are matched.

In paragraph "9" of the Office Action, the Office asserts that Neiman discloses a system-defined limit that anticipates claim 3 of the Application. Job termination during computation "(e.g., by reassignment of the node computer to a new service (step 1830) or by failure of the node computer 800)", Neiman p. 13, paragraph 0134)" is not a system-defined limit, it is simply

job failure. Failure to match a job with a worker is different from job failure. Neiman does not disclose how to "match" jobs with workers, and therefore cannot count how many times a match has been attempted or whether a system-defined limit has been reached. Neiman lacks three key structural elements of the EDQS invention: matching jobs with workers (Neiman's job-driven "allocation" is not job-driven matching since "allocation" is not described and is presumed to be simply waiting for the worker specified in a job request to become available), worker-driven match, and system-defined limits of attempts to match job and worker.

In paragraph "10" of the Office Action, the Office asserts that Neiman discloses a check environment process that anticipates claim 4 of the Application. "Allocate an appropriate amount of computing resources to particular jobs 182-1 to 182-N based on (1) the amount of resources allocated to a particular service and (2) the resource requirements of the jobs 182-1 to 182-N" (Neiman p. 7, paragraph 0075) is *routing* based on the *amount* of resources, not the *type* of resources (e.g., software version, rendering program, etc.) or a process that ranks workers based on meta-information (in Nieman) or job attributes and worker capabilities (in the EDQS invention). The EDQS check environment process compares the amount and type of resources (paragraphs 66 and 67 of the Application) and ranks workers. Neiman lacks this key structural element of the EDQS invention.

In paragraph "11" of the Office Action, the Office asserts that Neiman discloses a system-defined limit and reiterative processing that anticipates claim 5 of the Application. Re-queuing a failed job is not a system-defined limit, it is simply "re-allocating" a job after job failure. Failure to match a job and worker is different from job failure in Nieman paragraph 0093. Neiman does not "match" jobs with workers, does not rank workers, and therefore cannot count how many times a match has been attempted, whether a system-defined limit has been reached, or dispatch a job to the next ranked worker. Neiman lacks three key structural elements of the EDQS invention: matching jobs and workers (Neiman's job-driven "re-allocation" after job failure is not job-driven matching; a scheduling algorithm are not disclosed in Nieman), job-driven match that includes ranking workers, and system-defined limit of attempts to match a job with a worker. The terms, or function, of "sort" and "rank" are not disclosed in Nieman, and therefore Nieman cannot anticipate an invention that has sorting and ranking as key structural elements.

In paragraph "12" of the Office Action, the Office asserts that Neiman discloses a worker-driven job matching and job dispatch by aliasing, filtering, preparing an eligible job list, matching a worker with a job, and job dispatch to the worker that anticipates claim 6 of the Application. As explained connection with the discussion of claim 1 above, Nieman does not disclose "aliasing workers to nodes", but simply deploying node computers on a backbone. Neiman's "worker module" is analogous to a "job process" managed by an "executor" in the EDQS invention (Application, paragraph 56) and Neiman's "node" is a "worker" in the EDQS invention. Neiman does not alias worker modules or anything else. Neiman lacks the structural elements of nodes and workers aliased to nodes, as defined in the EDQS invention. Neiman's "queue 500" is not a "queuing system", but simply a job list. "Schedule jobs 182-1 to 182-N (which are associated with a particular calling application 180) with a worker 155 that resides on an available node computer 800 of the compute backbone 300" (Nieman, p. 4, paragraph 0054) is not filtering the eligible job list to match a job with a worker; Neiman discloses a job queue but now how jobs are matched to workers or, conversely and as relevant to claim 6, how workers are matched to jobs. "Schedules the jobs 182-1 to 182-N on available node computers 800-1 to 800-N" (Neiman pp. 11-12, paragraph 0122) is not a building an eligible worker list (available does not mean capable, i.e., eligible) and does not describe how existing jobs are matched to newly available workers. "Schedules the jobs 182-1 to 182-N on available node computers 800-1 to 800-N" (Neiman pp. 11 - 12, paragraph 0122) is not filtering jobs to produce an eligible job list; Nieman does not disclose filtering, sorting, or other matching method. "Determines availability of the node computers 800-1 to 800-N (step 1630), and schedules the jobs 182-1 to 182-N on available node computers 800-1 to 800-N" (Neiman pp. 11 - 12, paragraph 0122) is not selecting a job from the eligible job list; the description in Nieman is in connection with selecting a worker for a job (job-driven), not selecting a job for a worker (worker-driven) in claim 6. "Processing of a new reservation 2005 and/or a new request 2007" (Nieman p. 15, paragraph 0151) is not reserving or matching a newly available worker to a job (worker-driven) but matching a new job request to a worker (job-driven). Neiman does not disclose worker-driven allocation, only job-driven allocation, and does not disclose how jobs are matched with workers. Therefore Neiman lacks all the key structural element required for a *worker-driven* job dispatch as described in the EDQS invention.

In paragraph "13" of the Office Action, the Office asserts that Neiman discloses removal of a job from the eligible job list that anticipates claim 7 of the Application. Removing the worker from the "service allocated" (Nieman p. 9, paragraph 0093) is not removing a job from the eligible job list because Neiman does not construct an eligible job list or disclose a worker-driven dispatch. Nieman paragraph 0093 describes worker failure, not a newly available worker. Re-allocating a job is not removal of the job, it is survival of the job. Therefore Neiman lacks key structural elements, eligible job list and worker-driven dispatch, of the EDQS invention.

In paragraph "14" of the Office Action, the Office asserts that Neiman discloses a job sort after filtering reaches a system-defined limit that anticipates claim 8 of the Application. Re-queuing a failed job, as desribed in Nieman paragraph 0134, is not a system-defined limit, it is simply "re-allocating" a job after job failure. Failure to match a worker with a job is different from job failure. Neiman does not "match" workers with jobs, does not rank jobs, and therefore cannot count how many times a match has been attempted or whether a system-defined limit has been reached or dispatch a job to the next ranked worker. Neiman lacks three key structural elements of the EDQS invention: matching jobs and workers (Neiman's "re-allocation" after job failure is not worker-driven matching since "allocation" is not described), worker-driven match that includes ranking jobs and system-defined limit of attempts to match a worker with a job.

In paragraph "15" of the Office Action, the Office asserts that Neiman discloses a check environment process that anticipates claim 9 of the Application. The step of "compares job attributes with worker attributes" is not described in Nieman p. 7, paragraph 0075; the word "compare" does not occur in Nieman. Job attributes and matching job attributes with worker capabilities are not discussed in Nieman paragraph 0093. Removing the worker from the "service allocated" (Nieman p. 9, paragraph 0093) is not removing a job from the eligible job list because Neiman does not construct an eligible job list or disclose a worker-driven dispatch. Nieman paragraph 0093 describes worker failure or reassignment, not a newly available worker. "Allocate an appropriate amount of computing resources to particular jobs 182-1 to 182-N based on (1) the amount of resources allocated to a particular service and (2) the resource requirements of the jobs 182-1 to 182-N" (Neiman p. 7, paragraph 0075) is *routing* based on the quantitative amount of resources, not non-quantitative *type* of resources (e.g., software version, rendering program, etc.) or a process that ranks jobs based job attributes (in the EDQS invention). The EDQS check environment process compares job attributes (paragraphs 66 and 67 of the

Application) and ranks jobs in a worker-driven dispatch. Neiman lacks this key structural element of the EDQS invention. Therefore Neiman lacks key structural elements, eligible job list and worker-driven dispatch, of the EDQS invention.

In paragraph "16" of the Office Action, the Office asserts that Neiman discloses a system-defined limit that anticipates claim 10 of the Application. Re-queuing a failed job is not a system-defined limit, it is simply "re-allocating" a job after worker failure. Nieman paragraph 0093 does not discuss worker rejection of a job, but worker failure or being unavailable for allocation. Neiman does not "match" workers with jobs, does not rank jobs, and therefore cannot count how many times a match has been attempted, whether a system-defined limit has been reached, or dispatch a worker to the next ranked job. Neiman lacks key structural elements of the EDQS invention: worker-driven matching of jobs with workers (Neiman's job-driven "re-allocation" after job failure is not worker-driven matching) and system-defined limit of attempts to match a worker with a job. Nieman paragraph 0134 does not address sorting, a job list, or ranking. The terms, or function, of "sort" and "rank" are not used or disclosed in Nieman, and therefore Nieman cannot anticipate an invention that has sorting and ranking as key structural elements.

Claim Rejections - 35 USC § 103

In paragraph "17" of the Office Action, the Office rejected claim 11 - 20 under 35 U.S.C. 103(a) as being unpatentable over Neiman in view of U.S. Patent No. 6,182,110 to Barroux.

Neiman has been disqualified as a reference as of Neiman's filing date by the Declaration of Anthony Higa. Without Neiman, Barroux cannot render claims 11 – 20 obvious.

Notwithstanding the disqualification of Nieman, and without waiver of such disqualification of Nieman, the following comparison of Barroux versus the EDQS invention establishes that Nieman in view of Barroux does not render claims 11 - 20 obvious.

In paragraph "20" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches a queuing system that renders claim 11 of the Application obvious. The primary inventive steps of the EDQS invention are not the use of clients, supervisors, and workers, but the method by which *jobs and workers are matched*, how workers are managed upon change of state, how dependencies are enforced, and how heterogenous platforms communicate over a network *without building unique messages for the API* on each type of user and worker platform

and *without the calling application(or its API) knowing the specific identity of the worker* that will execute a job. Neiman not disclose how jobs and workers are matched, how workers are managed upon change of state (other than a failed job being sent to the back of the queue), how dependencies are enforced, and how heterogeneous platforms communicate over a network without building unique messages for the API on the specified worker platform. As explained above, the prior art scheduling system of Neiman is not "event-driven", as that term is defined in the Application. Neiman is job-driven, and the method of matching job and worker *is not defined, and therefore is not enabled* except perhaps as an existing art "periodic sort" method.

Barroux teaches a basic reservation system, not a scheduling system. Barroux "allows a user to schedule node-specific tasks across the network without specifying particular times for each node" *only if the particular time was not previously reserved* (or in Barroux's terms, in the absence of "negative schedule information", see discussion of Barroux' Figure 3). Barroux' "task scheduler" is a three-step Boolean test for whether a previously scheduled job exists for a given time of day and duration: time slot availability is tested at the node, subnet, and network levels. A Boolean test for a time slot is not a scheduling system, it is the most basic type of reservation system, which Barroux chooses to call a "task scheduler". "Event" in Barroux *is not defined.* Barroux states, in the discussion of Figure 3, that "Event handler 312 sends a Clock Timer message to clock process 310 every sixty seconds." An "event" in Barroux, by implication, is a simply a timing pulse.

Paragraph [112] of the Application states, "The term "event" technically means the event defined in the relevant evtag. The format of each evtag is as follows:

<evname>-<evtype>-<evcontext>[-<evextra1>][-<evextra2>]...[-<evextraN>]"

The dictionary meaning of "event" is not "<evname>-<evtype>-<evcontext>[-<evextra1>][-<evextra2>]...[-<evextraN>]" and nothing more than "timing pulse" can be read into Barroux' "event". Moreover, Barroux does not include a component selected from the group comprising EDQS messaging architecture, EDQS event-driven dispatch, EDQS event/callback architecture. EDQS job type data architecture, and the EDQS domain specific language. Scheduling according to time of day, if there are no prior reservations for that time of day, is not EDQS messaging architecture, EDQS event-driven dispatch, EDQS event/callback architecture, EDQS job type data architecture, or the EDQS domain specific language. Therefore, Neiman in view of Barroux cannot render claim 11 obvious.

In paragraph "21" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches a job type data architecture that renders claim 12 of the Application obvious. Job type element is defined paragraph [89] in the Application. Neiman's list of *optional* meta-information in paragraphs 44 and 45 do not include the *minimum required* data of a job type element, which are:

name of job type element ("descriptor" datum, the collection of meta-information of Neiman paragraph 45 is not named),

name of execution file for processing the job ("executor" datum, which is the name of executable code, not identification of a particular worker)

name of the submission dialogue file ("GUI name" datum; Neiman uses a GUI, but the name of the GUI is not meta-information),

name of job iconic representation file ("icon" datum; a desktop icon is not a set of "service type checkboxes" described by Barroux),

submission command line, including command line arguments ("commander" datum; the queue or list of Nieman is not a submission command line),

binding scripts (the de-installation process of Barroux is not a binding script),

names of associated libraries (Nieman's four layer software architecture is not the names of associated libraries) and

name of index file (Barroux' list of database fields is not the name of an index file).

Concepts and unrelated elements scattered through Nieman and Barroux cannot be retrospectively integrated into the mandatory components of a job type element as defined in paragraph [89] in the Application. Moreover, job type elements are only one component of the job type architecture of the EDQS invention. Neither Neiman nor Barroux disclose a job type element or job type architecture, or anything analogous to these EDQS structural elements, and therefore cannot render claim 13 obvious.

In paragraph "22" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches a job type directory that contains the job type element, together with code files identified in the job type element that render claim 13 of the Application obvious. However, neither Neiman nor Barroux disclose a job type element (as explained in the discussion of claim 12) or include executable code in anything analogous to a job type element, and therefore cannot render claim 13 obvious.

In paragraph "23" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches the EDQS messaging architecture, including messages written in the EDQS domain specific language [col. 4, lines 36 - 65, col. 6, lines 36 - 45 of Barroux] and therefore renders claim 14 of the Application obvious. Barroux, col. 4, lines 36 - 65, col. 6 and lines 36 – 45, describe the clock process and a list of messages, but does not disclose a messaging architecture or a domain specific language, as defined in the Application, or any other type of messaging architecture or language. Neiman mentions off-the-shelf programming lanuages, e.g., Smalltalk and C. Barroux does not contain the word "language" or equivalent word. Neither Nieman nor Barroux describe a domain specific language, as defined in the Application, so Neiman in view of Barroux cannot render claim 14 obvious.

In paragraph "24" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches that the domain specific language uses a message generator that converts platform-specific commands into a data structure that can be interpreted by other types of platforms [pp. 5-6, paragraph 0066] and therefore renders claim 15 of the Application obvious. Neiman mentions off-the-shelf programming languages, e.g., Smalltalk and C. Barroux does not contain the word "language" or equivalent word. Neither Nieman nor Barroux describe a domain specific language, as defined in the Application (paragraph [102 – 140]), so Neiman in view of Barroux cannot render claim 15 obvious.

In paragraph "25" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches the event/callback architecture and therefore renders claim 16 of the Application obvious. The event/callback architecture and triggers is defined in paragraph [110] of the Application. Barroux col. 4, lines 36 - 65, col. 6, lines 36 – 45, describe the clock process, but not the event/callback architecture described in paragraphs 102-140 of the Application, and specifically does not describe anything analogous to the definition of triggers in paragraphs [120 - 123] of the Application. Neither Nieman nor Barroux describe triggers, an event/callback, or an equivalent, as defined in the Application (paragraph 102 - 140), so Neiman in view of Barroux cannot render claim 16 obvious.

In paragraph "26" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches callback code and therefore renders claim 17 of the Application obvious. Barroux col. 4, lines 36 - 65, col. 6, lines 36 – 45, describe the clock process, but not the event/callback architecture described in paragraphs 102-140 of the Application, and specifically

the definition of callback code in paragraphs [120 – 123]. Callback code is the executable code that associated with a trigger. Neither Nieman nor Barroux describe triggers or callback code as defined in the Application (paragraph 126), or an equivalent, so Neiman in view of Barroux cannot render claim 16 obvious.

In paragraph "27" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches the callback code causes a worker-driven dispatch and therefore renders claim 18 of the Application obvious. As explained in connection with claim 17 above, neither Nieman nor Barroux describes callback code, and as explained in connection with claims 3, 6, 7, 8, 9, and 10 above, neither Nieman nor Barroux describes a worker-driven dispatch, so Neiman in view of Barroux cannot render claim 17 obvious.

In paragraph "28" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches the trigger evaluates Boolean expressions of event states of one or more jobs in a process group and therefore renders claim 19 of the Application obvious. Barroux col. 9, lines 52 – 65 describes network interface tables ("e.g., Ethernet, token ring, starlan, or fddi, a physical address (MAC) address of the network interface, an IP address of the network interface as retrieved by SNMP probe system 214, a device name of the interface unit, and a description of the interface unit."). This network interface data is part of the asset survey of Barroux. As explained in connection with claim 17 and 18 above, neither Nieman nor Barroux describes callback code, and network interface tables have nothing to do with evaluation of Boolean expressions, so Neiman in view of Barroux cannot render claim 18 obvious.

In paragraph "29" of the Office Action, the Office asserts that Neiman as modified by Barroux teaches the event/callback architecture uses evname, evtype, and evcontext variables to define an event [col. 6, lines 26 - 37 of Barroux] and therefore renders claim 20 of the Application obvious.

"Event" in Barroux means a specified time of the day. In contrast, paragraph [112] of the Application states, "The term "event" technically means the event defined in the relevant evtag. The format of each evtag is as follows:

<evname>-<evtype>-<evcontext>[-<evextra1>][-<evextra2>]…[-<evextraN>]"

The terms "evname," "evtype," and "evcontext" variables are carefully defined and exemplified in the Application. Neiman and Barroux have no equivalents of the evname, evtype, and evcontext variables and therefore cannot render claim 20 obvious.

The obviousness rejection cannot stand because the combination of the cited references is legally improper

The need for specificity pervades the authority of the Office to reject claims based on obviousness and combined references. See, *e.g., In re Kotzab*, 217 F.3d 1365, 1371, 55 USPQ2d 1313, 1317 (Fed. Cir. 2000) ("particular findings must be made as to the reason the skilled artisan, with no knowledge of the claimed invention, would have selected these components for combination in the manner claimed"); *In re Rouffet*, 149 F.3d 1350, 1359, 47 USPQ2d 1453, 1459 (Fed. Cir. 1998) ("even when the level of skill in the art is high, the Board must identify specifically the principle, known to one of ordinary skill, that suggests the claimed combination. In other words, the Board must explain the reasons one of ordinary skill in the art would have been motivated to select the references and to combine them to render the claimed invention obvious."); *In re Fritch*, 972 F.2d 1260, 1265, 23 USPQ2d 1780, 1783 (Fed. Cir. 1992) (the examiner can satisfy the burden of showing obviousness of the combination "only by showing some objective teaching in the prior art or that knowledge generally available to one of ordinary skill in the art would lead that individual to combine the relevant teachings of the references").

The Office cannot rely on conclusory statements or a piece meal joining of the references ("A person having ordinary skill in the art would have been motivated to use such modification because it would increase real estate business volume transactions."). *In re Sang Su Lee,* 277 F.3d 1338, 1342 (Fed. Cir. 2002) ("The examiner's conclusory statements that 'the demonstration mode is just a programmable feature which can be used in many different device[s] ... by adding the proper programming software' and that 'another motivation would be that the automatic demonstration mode is user friendly and it functions as a tutorial' do not adequately address the issue of motivation to combine. This factual question of motivation is material to patentability, and could not be resolved on subjective belief and unknown authority. It is improper, in determining whether a person of ordinary skill would have been led to this combination of references, simply to '[use] that which the inventor taught against its teacher'." *W.L. Gore v. Garlock, Inc.,* 721 F.2d 1540, 1553, 220 USPQ 303, 312-13 (Fed. Cir. 1983)).

The references cited in the Office Action do not contain sufficient "teaching or suggestion to make the claimed combination and the reasonable expectation of success," *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991). Without such teaching and expectation

of success, express or clearly implied in the references, the Office has failed to establish *prima facie* obviousness. The rejection also lacks the necessary connection of the various elements required by *In re Kotzab*, 217 F.3d 1365, 55 USPQ2d 1313 (Fed. Cir. 2000).

Applicants have tried in vain to identify specific portions of the two cited references, or even implications in them, that would suggest the motivation or suggestion to combine them together as the Office Action has done. There is no express or implied suggestion or motivation in the cited references; the Office has not identified any express or implied suggestion or motivation to combine the references and has failed to meet the requirements of a *prima facie* showing of obviousness.

Conclusion

Applicants respectfully submit that the rejections are improper and should be withdrawn in light of the Remarks and Amendments above. All pending claims are allowable, and Applicants respectfully solicit an early notice to that effect. The Office is invited to contact Applicants' undersigned representative if there are any questions relating to the subject application.

Respectfully submitted,

Date: 29 March 2007

George E. Darby
Registration No. 44,053
Telephone: (808) 626-1300
Facsimile: (808) 626-1350

USPTO Customer Number 25668

Paradise Patent Services, Inc.
P.O. Box 893010
Mililani, HI 96789-3010